2015

Smart Laboratory Instrument Control Framework

Kevin Scott Kennedy

SMART LABORATORY INSTRUMENT CONTROL FRAMEWORK

by

Kevin Scott Kennedy

Thesis submitted in partial fulfillment
of the requirements for the degree

of

DEPARTMENTAL HONORS

in

Electrical Engineering
in the Department of Electrical and Computer Engineering

Approved:

Thesis/Project Advisor	**Departmental Honors Advisor**
Dr. Don Cripps	Dr. V. Dean Adams

Director of Honors Program
Dr. Kristine Miller

UTAH STATE UNIVERSITY
Logan, UT

Spring 2015

Smart Laboratory Instrument Control Framework

Kevin Kennedy, Justin Jonas, Robbie Schaap

May 12, 2015

Contents

List of Figures

1. Abstract

The collection of experimental results in power electronics research requires the coordination of a wide variety of laboratory instrument. Depending on the type of results required, each data point can take a few minutes to collect. This process becomes time consuming when a large number of data points is needed and detracts from long-term research objectives. While there are industry standards that specify communication with these devices, many equipment vendors elect not to follow the standards and use their own protocols for communication. This project provides a framework for developing a smart and connected lab. This includes hardware and software for interacting with all lab instruments, regardless of the vendor. The framework is expandable to accommodate future instruments while still being simple enough to learn quickly. With minimal training, students can create graphical user interfaces and automation scripts that will greatly simplify interacting with research projects.

2. Introduction

2.1. Background

The Utah State University Power Electronics Lab (UPEL) was founded in 2012 by Dr. Regan Zane, USTAR Professor in the ECE Department at USU. The lab is part of a program with Utah State Technology and Research (USTAR) to build world-class research centers in energy and power. The challenges addressed by UPEL span a wide range of topics with direct impact to society, including:

- Improving the efficiency, reducing weight and extending lifetime and range of electric vehicles
- Improving the battery management systems to extend the life of batteries in devices ranging from smart phones to an electric bus improving the efficiency and performance of computing, lighting, and power distribution in data centers and commercial buildings
- Enabling autonomous operation of miniature wireless sensors
- Integration of renewable energy sources, energy storage, and intelligent controls to improve reliability and capability in AC and DC micro-grids
- Development of high performance power conversion for military and aerospace applications.

2.2. Problem

Research laboratories utilize a wide variety of instruments and equipment to perform a wide variety of tasks from data collection to experimental verification. These instruments are often extremely specialized and expensive, so the training for researchers can be tedious. Some equipment vendors provide computer software to ease the use of their product, but when equipment from multiple vendors are used together, juggling the software packages can become cumbersome.

Fortunately, equipment vendors generally provide a remote control interface that uses an industry standard protocol. The use of this interface requires the development of a custom computer application, a skill that students generally have limited exposure to.

It is very common for labs and corporations to develop their own in-house framework for controlling instruments, but they are rarely released for general usage. Our project will be open-source software that others can use and continue to develop.

2.3. Solution

We designed a smart laboratory instrument control and automation solution that integrated equipment in the lab with control software on student workstations. We will leverage the remote control capability of equipment in the lab to create a unified interface for each instrument, regardless of

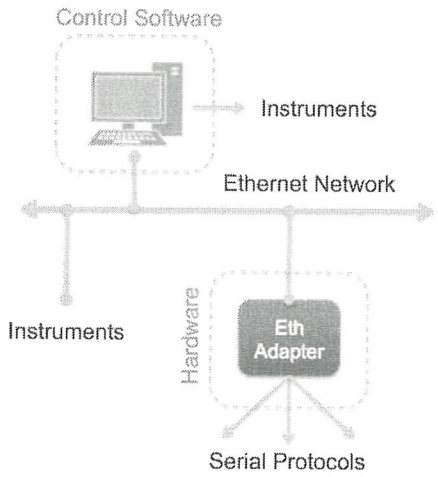


Figure 2.1.: System Overview

the equipment vendor.

The project has two major components, PC control software to manage communication with instruments and an Ethernet adapter module that facilitates connection to equipment that lacks an Ethernet port.

2.3.1. Control Software

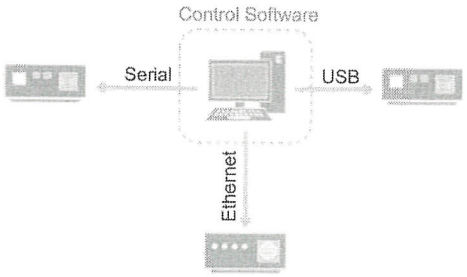


Figure 2.2.: Control Software Overview

The primary design objectives for the control software were modularity and extensibility. Modularity means that the functionality for each instrument is developed as a module and is completely isolated from other instrument modules. This makes it easy to develop new modules for new instruments. The project is extensible because it is easy to add new functionality to the framework using a simple plugin architecture. In this way, students are able to easily develop new scripts and interfaces for future equipment without unreasonable development time.

The software is compatible with a wide variety of instruments and protocols to ensure maximum usefulness. Furthermore, the control software provides for long-term data collection. The data collection feature will facilitate tests that run for long periods of time and need to collect large amounts of data. This feature, in combination with an easy-to-use script interface, will allow students to develop automation applications with ease.

2.3.2. Ethernet Adapter Module

This module acts as a mediator between the instruments and the network, thus allowing any instruments with a serial port to be connected to and controlled via an Ethernet network. Booster packs create the possibility for connection to many different instruments including RS-232, RS-485, and CAN.

2.4. Overview

Our solution improved lab safety, facilitates long-term testing through automation and allows for robust bench-top control of lab instruments and projects. With minimal training, students are able to create graphical user interfaces and automation scripts that will greatly simplify research projects now, and in the future.

3. Requirements

This section of the report will detail the requirements for the Ethernet adapter hardware module. Many of these requirements were revised during the design and development phases of the project. Justification for these changes are found in the Methods section of this report.

3.1. Non-Functional Requirements

3.1.1. Environment

3.1.1.1. The device shall be able to operate in the temperature range of $0° - 60°C$

3.1.1.2. The device enclosure shall have at least three mounting points to secure the Printed Circuit Board (PCB)

3.1.2. Interface

3.1.2.1. The device shall have an LCD display to report error conditions to the user

3.1.2.2. The device shall have an RJ-45 connector for Ethernet communication

3.1.2.3. The device shall have three (3) DB-9 connectors for communication with devices using the CAN, RS-232 and RS-485 protocols

3.1.2.4. The device shall have a header to connect a JTAG debugger. The header shall be a shrouded 10-pin (5x2) header with 1.27mm (0.05") pitch

3.1.2.5. The device shall have a USB connector with virtual serial port capability for debugging

3.1.2.6. The device shall have at least four (4) General Purpose I/O pins capable of withstanding at least 5.5 volts. These pins may be multiplexed with communication ports that do not have a specific connector specified

3.2. Functional Requirements

3.2.1. Communications Interface: CAN

3.2.1.1. The device shall be able to communicate with devices which implement the CAN Protocol (ISO 11898)

3.2.1.2. The device shall include the CAN bus-termination resistor between the CAN-H and CAN-L wires according to the CAN specification

3.2.1.3. The device shall be able to relay all communications from a CAN Bus device to any other device on the Ethernet network

3.2.1.4. The device shall require a destination CAN ID for all outbound communications

3.2.2. Communications Interface: RS-485

3.2.2.1. The device shall be able to communicate with devices which implement the RS-485 Protocol

3.2.2.2. The device shall be able to send arbitrary data onto the RS-485 bus

3.2.2.3. The device shall be able to relay all communications from the RS-485 bus to any other device on the Ethernet network

3.2.3. Communications Interface: RS-232

3.2.3.1. The device shall be able to communicate with devices which implement the RS-232 Protocol

3.2.3.2. The device shall be able to relay all communications from a RS-232 device to any other device on the Ethernet network

3.2.4. Communications Interface: Serial Peripheral Interface (SPI) Bus

3.2.4.1. The device shall be able to act as an SPI Master device

3.2.4.2. The device shall have one (1) chip-select pin, one (1) MISO (Master-in Slave-out) pin, one (1) MOSI (Master-out Slave-in) pin and one (1) Clock pin

3.2.4.3. The device shall support all SPI modes (combinations of clock polarity and phase)

3.2.5. Communications Interface: Inter-Integrated Circuit (I2C) Bus

3.2.5.1. The device shall be able to communicate with devices which implement the I2C Protocol

3.2.5.2. The device shall be able to act as an I2C Master device

3.2.5.3. The device shall support both the 7-bit and the 10-bit addressing schemes

4. Methods

4.1. Software Design

4.1.1. Objectives

In order to be effective, our solution must meet the following criteria:

- The software must be intuitive and easy to use
- The software must be compatible with all instruments in the lab, with the ability to add more instruments in the future as needed
- All documentation should be clear with examples for developing additional drivers for future devices

4.1.2. Architecture

The control software portion of this project was developed in Python. The decision to use Python as the language for the framework was motivated primarily by these reasons:

- Python is popular, easy to learn and used by many companies in the engineering and scientific industry
- Python has strong object-oriented capabilities
- Python has an extensive set of libraries that can be leveraged
- Python is free and open-source

The control software uses a Presentation-Abstraction-Control [1] architectural pattern to separate the instrument control functionality from the way the instrument is presented to the user. which separates the low-level system calls for communication from the instrument drivers. By using this design approach, core program functionality is isolated into smaller chunks in order to keep errors from propagating throughout the entire program. This was done to make the development of new instruments easy for users without extensive object-oriented programming experience.

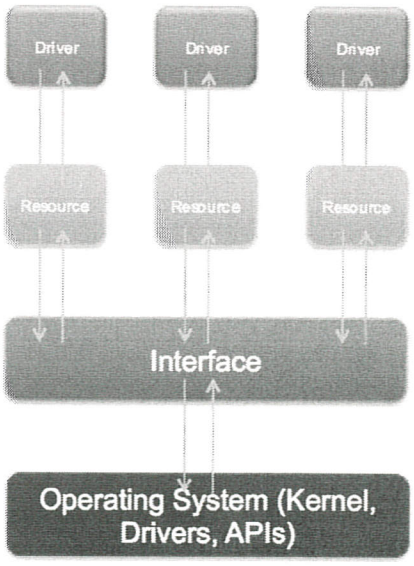


Figure 4.1.: Software Architecture

Interfaces

Interfaces handle the low-level operating system interactions. This includes interaction with hardware in the computer to communicate with the instruments.

Interfaces typically leverage third-party drivers supplied by vendors to handle communication. An example of this kind of interface is the National Instruments Virtual Instrument Software Architecture (NI-VISA). For a list of supported interfaces, see section 4.1.3.

Resources

Resources are objects created by the interface that represent a physical device connected to the system. The resource does not necessarily need to be connected to the user's computer, but could be connected to a remote computer.

Drivers

Drivers contain all of the code and commands needed to interact with a specific instrument. They are loaded atop a resource to give a device connected to the system an identity (vendor, model, serial number) and a set of commands the user can use to control the instrument. For a list of supported instruments, see section 4.1.4.

4.1.3. Supported Interfaces

- Virtual Instrument Software Architecture (VISA)
- USB
- RS-232 (Serial)
- GPIB
- Ethernet (VXI)
- Instrument Control Protocol

4.1.4. Supported Instruments

Vendor	Model	Type
Agilent Technologies	34411A	Digital Multimeter
BK Precision	9100 Series	DC Power Supplies
BK Precision	XLN Series	DC Power Supplies
BK Precision	8500 Series	DC Loads
Chroma	62000P Series	DC Power Supplies
Regatron	Grid-Tie Source/Sink (GSS)	AC/DC Power Supply
TDI Power	XBL Series	DC Loads
Tektronix	DPO 2000 Series	Digital Phosphor Oscilloscopes
Tektronix	MSO 2000 Series	Mixed Signal Oscilloscopes
Tektronix	MSO 5000 Series	Mixed Signal Oscilloscopes

4.2. Hardware Design

4.2.1. Preliminary Design

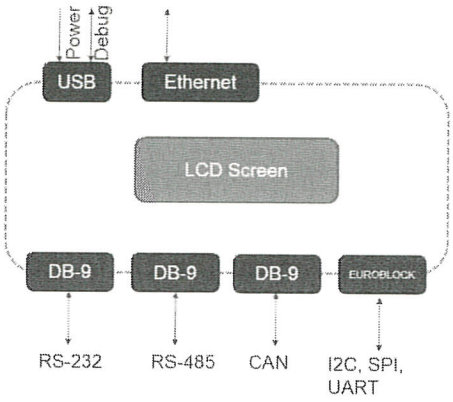


Figure 4.2.: Hardware Overview

Given the scope of our project and the connectors we planned to use, we decided it would be best to design a custom PCB. We used Altium Designer to capture the electrical schematics and PCB layout. The connectors and protocols were specified in the requirements. Schematics can be found in the appendix.

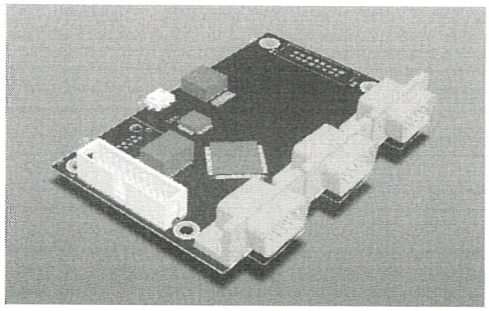


Figure 4.3.: PCB 3D Rendering

Microcontroller

The project requirements dictated the need for an Ethernet port for communication with the local network. The desire to keep costs low constrained our decision to use a microcontroller for the computational component of our project. A few different microcontrollers were considered because of their integrated Ethernet controllers. The list of parts considered can be found in table 4.1.

In the end, we decided to use the Texas Instruments TM4C1294NCPDT part for the following reasons:

- Integrated Ethernet PHY eliminated the need for an external component

Vendor	Part	Clock Speed	CPU	Flash	Cost
Texas Instruments	TM4C1294NCPDT	120 MHz	ARM Cortex-M4F	1024Kb	15.00
Atmel	ATSAM4E8CA	120 MHz	ARM Cortex-M4	512Kb	9.73
Atmel	ATSAM3X8E	84 MHz	ARM Cortex-M3	512Kb	12.28

Table 4.1.: Microcontroller Selection Matrix

- Integrated CAN controller eliminated the need for an external controller
- TivaWare libraries provide boiler-plate code for rapid development
- Access to cheap development boards would allow concurrent development of hardware and software

LCD Screen

The requirements specified an LCD screen to report any error conditions to the user. During normal operation, there is very little information to report to the user, so a simple 16x2 character LCD was selected as shown in figure 4.4.

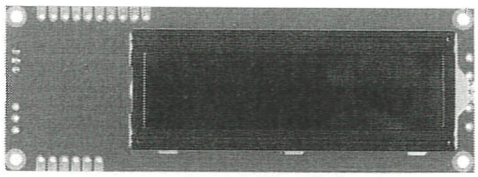


Figure 4.4.: 16x2 Character LCD Screen

Enclosure

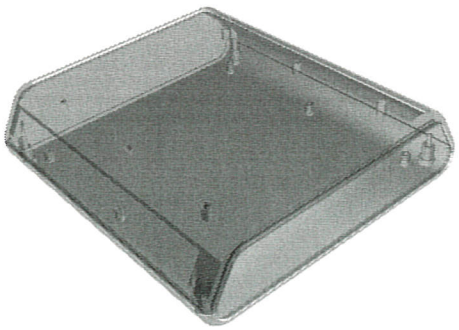


Figure 4.5.: Polycase JB-65B Enclosure

Electro-static Discharge (ESD) is a serious concern for electronics designers, so we made sure to include precautions to prevent damage to our board. One of these precautions was to enclose the board. We selected an injection molded ABS plastic case from Polycase. Each case is $9.85.

4.2.2. Prototype

After the PCB was designed and ready for production, we discovered that the pitch between the pins on the microcontroller were so fine that it would cost us much more than we expected for a prototype device. The design could be cost effective if manufactured in high volume, but small quantity prototypes were well outside of our budget.

In light of this discovery, we decided to use the Texas Instruments TM4C1294 Connected LaunchPad Evaluation Kit(LaunchPad) (Figure 4.6) to demonstrate a proof-of-concept prototype of our design. The LaunchPad uses the same microcontroller that we were planning to use, so the code would be portable to the final design. Instead of having all of the connectors on one board, we would design a series of add-on boards that could be swapped out. The prototype allowed us to demonstrate our design without incurring high development costs. The prototype can be seen in figure 4.7 with the BoosterPack add-on modules for each of the connectors and protocols. We made BoosterPack for CAN, RS-232, and RS-485 communication protocols. These booster packs were designed in Altium and manufactured to fit on the LaunchPad. These protocols were selected because these could be the most useful to those in the power lab who might use this prototype.

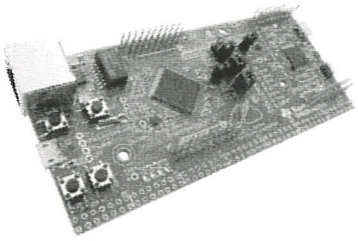


Figure 4.6.: TI TM4C1294 Connected LaunchPad

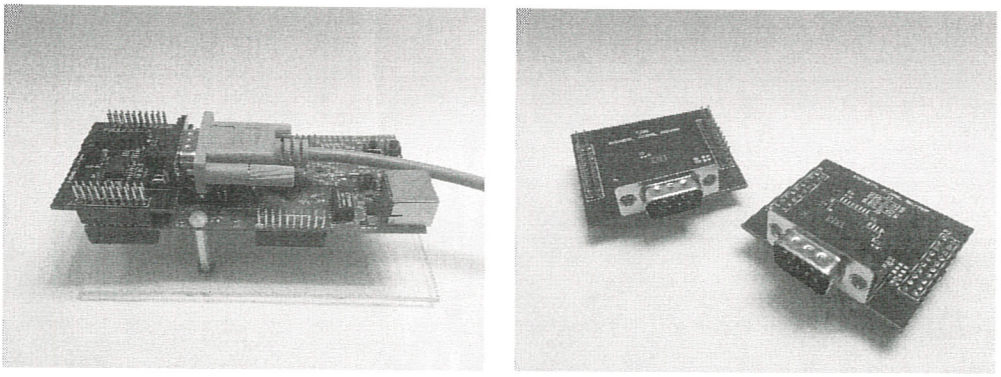


Figure 4.7.: Prototype and BoosterPacks

5. Results

5.1. Requirements Validation

This section will discuss the validation and testing of the requirements set out in section 3. The hardware and software testing will be described at a high level of functionality to show that the end product meets the requirements as a whole.

5.1.1. Prototype Hardware Test

As stated above, the prototype hardware was designed to be a proof-of-concept for the initial hardware design. The BoosterPack that was tested was the RS-232 because this is the most common communication protocol for lab instruments.

We tested the functionality by connecting a multimeter to the booster pack via an RS-232 cable. The LaunchPad was then connected to the local network via the Ethernet port. The firmware acted as a pass through to all commands given from the software and responses given by the instrument. This configuration enabled the software to control an instrument through the local network.

5.1.2. Software Test

The software was designed to be able to control many different instruments that might be used in the power lab. Each device was connected to a computer and then had each function in the device API tested. Below is a table that shows which instruments were designed and if they were tested.

Vendor	Model	Type	Test?
Agilent Technologies	34411A	Digital Multimeter	✓
BK Precision	9100 Series	DC Power Supplies	✓
BK Precision	XLN Series	DC Power Supplies	✓
BK Precision	8500 Series	DC Loads	✓
Chroma	62000P Series	DC Power Supplies	✓
Regatron	Grid-Tie Source/Sink (GSS)	AC/DC Power Supply	✓
TDI Power	XBL Series	DC Loads	✓
Tektronix	DPO 2000 Series	Digital Phosphor Oscilloscopes	✓
Tektronix	MSO 2000 Series	Mixed Signal Oscilloscopes	✓
Tektronix	MSO 5000 Series	Mixed Signal Oscilloscopes	✓

One of the main goals of this project was to make sure that the software is used after the project has been completed. In order to make sure this happened we tested the usability of the user interfaces that were created for each class of instrument and for testing scripts.

5.1.3. Case Study

A project came up in the USU Power Electronics Lab that needed to use lab automation. The project required testing a large quantity of hardware units using multiple instruments. By hand, this process took as long as 20 minutes. Users who had never before used the software or documentation had the opportunity to write their own scripts to test the hardware and shortened the process to less than 5 minutes. Figure 5.1 shows the user interface developed for this lab project.

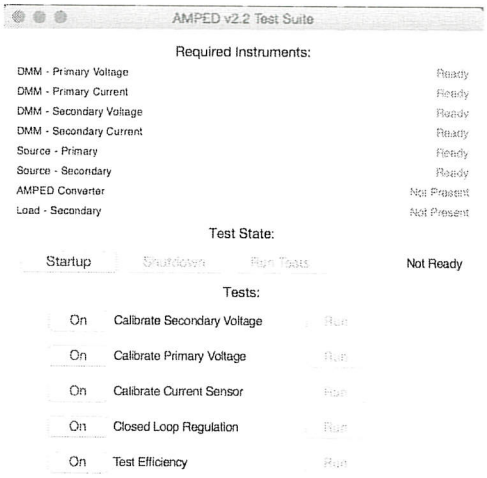


Figure 5.1.: AMPED Test Suite Interface

5.2. Discussion of Results

This section will discuss the results of the tests and the implications of the results.

5.2.1. Hardware

After testing the hardware module it successfully connected the multimeter and the software through the network. As defined in the requirements, there are many other communication protocols that would be useful. This test showed a proof of concept that the originally designed hardware module would work if manufactured for other protocols as well. The hardware module would help for specific situations like maybe a lab is far from a workstation. The initial hardware module would be a great help for labs with these types of situations.

5.2.2. Software

The first test described above for each of the instruments shows that the software does support most of the instruments in the lab. The software also supports adding new instruments in a simple way. The project documentation includes detailed instructions about this process.

This documentation can help users to run scripts for specific instruments, make drivers for new instruments, or combine instruments into one project running test scripts. This helps to fulfill

the goal of having this project be used after it is finished. All of the documentation will be online when the project is released as open-source. Figure 5.2 shows an example of the documentation.

Figure 5.2.: Online Software Documentation

The next test described was when we had an actual lab project that needed to use the software and documentation. The users of the software were writing and debugging their own scripts during the process. These two tests show that the software works for many of the instruments in the lab and that it can be used later on for different projects in the power lab.

5.3. Costs

The following project costs are tabulated for a single module irrespective of the costs that will be associated with building prototypes. This cost estimate does not include engineering time.

Part Number	Description	Cost
JB-65	Injection Mold Plastic Enclosure with cutouts	9.85
N/A	USB AC Power Adapter	7.85
TM4C129NCPDT	TI Tiva ARM Cortex-M4 1MB Flash	17.75
LMZ10503TZE-ADJ	3A SimpleSwitcher Power Regulator	8.44
ADM3053	CAN Transceiver	11.77
MAX3232EIDWR	RS-232 Line Driver	0.88
ADM3485ARZ	RS-485 Transceiver	2.82
HX1198FNL	Ethernet Magnetics	1.00
	DB-9 Through-hole (x2)	1.50
	Terminal Block (2x 4 Position)	2.12
	RJ-45 (Ethernet) Jack	2.60
	Power Supply Connector	1.00
	Printed Circuit Board	35.00
	Total Cost	**102.58**

Table 5.1.: Simplified Bill of Materials

6. Conclusion

6.1. Discussion

Automating and controlling lab instruments is not a new concept or idea. It is common for engineers and scientists in industry to write scripts and applications to run tests. There are even commercial software suites like National Instruments LabVIEW that can be used to develop automation scripts. These solutions are often criticized for being difficult to learn, and cumbersome to operate. Our solution provides a framework for others to build on that is:

- Developed in an industry-standard programming language
- Intuitive and easy to use
- Extensively documented
- Open-Source
- Expandable for the future

One of the metrics for the success of our project will be how much it is adopted by the Power Electronics lab and the Open-Source community. That is why we helped to develop some example applications for real projects in the lab that helped to boost productivity.

6.2. Lessons Learned

One of the most important aspects of the project was getting feedback from the end-users. Though we were developing a software solution to solve a general problem, it was important for us to keep in constant contact with those who would be using the software in the future after we all have graduated.

6.3. Future Work

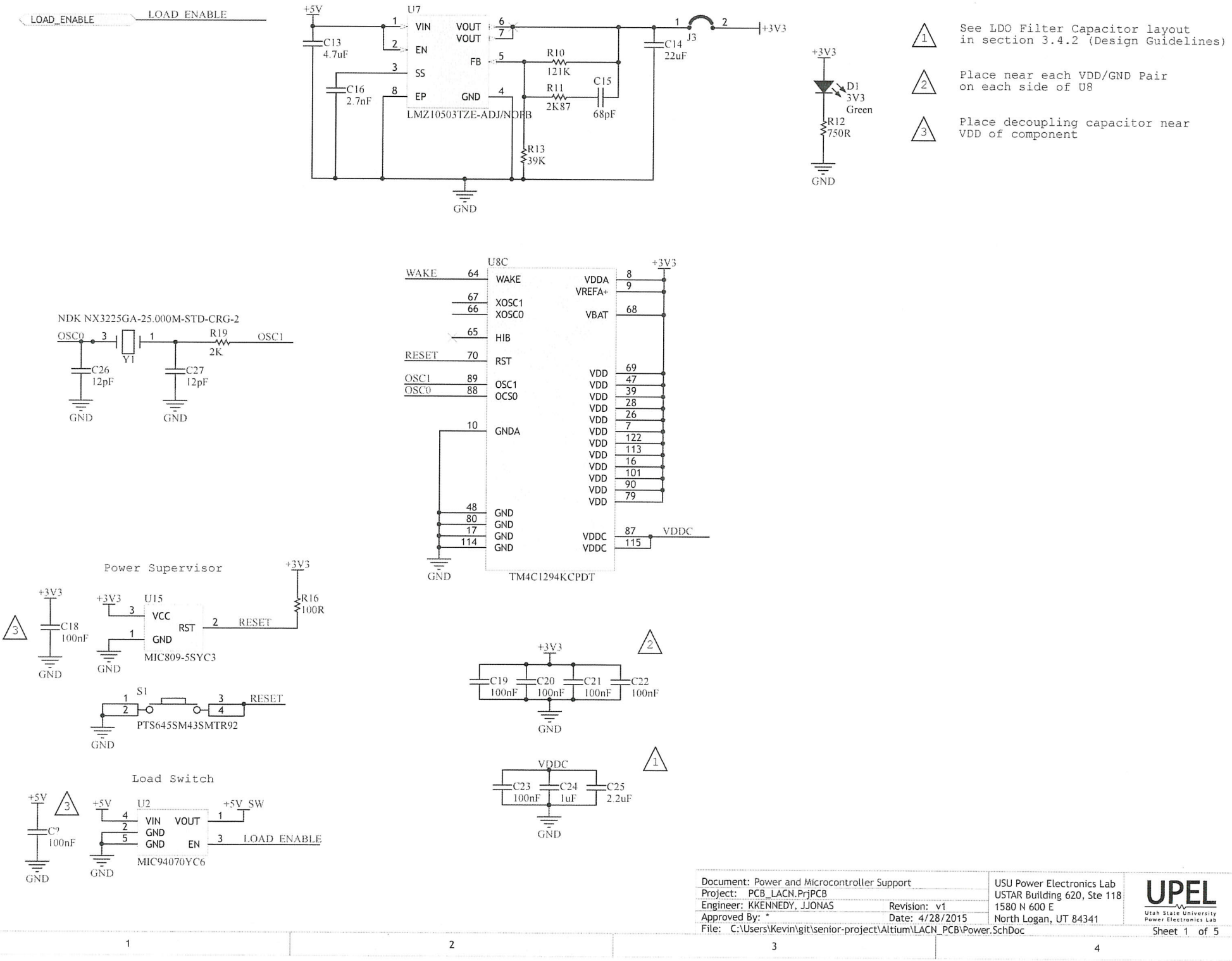
Many people, labs and organizations have shown an interest in the software the we have developed for instrument automation. As was said before, this project will become open-source so that it can continue to be developed and used.

Bibliography

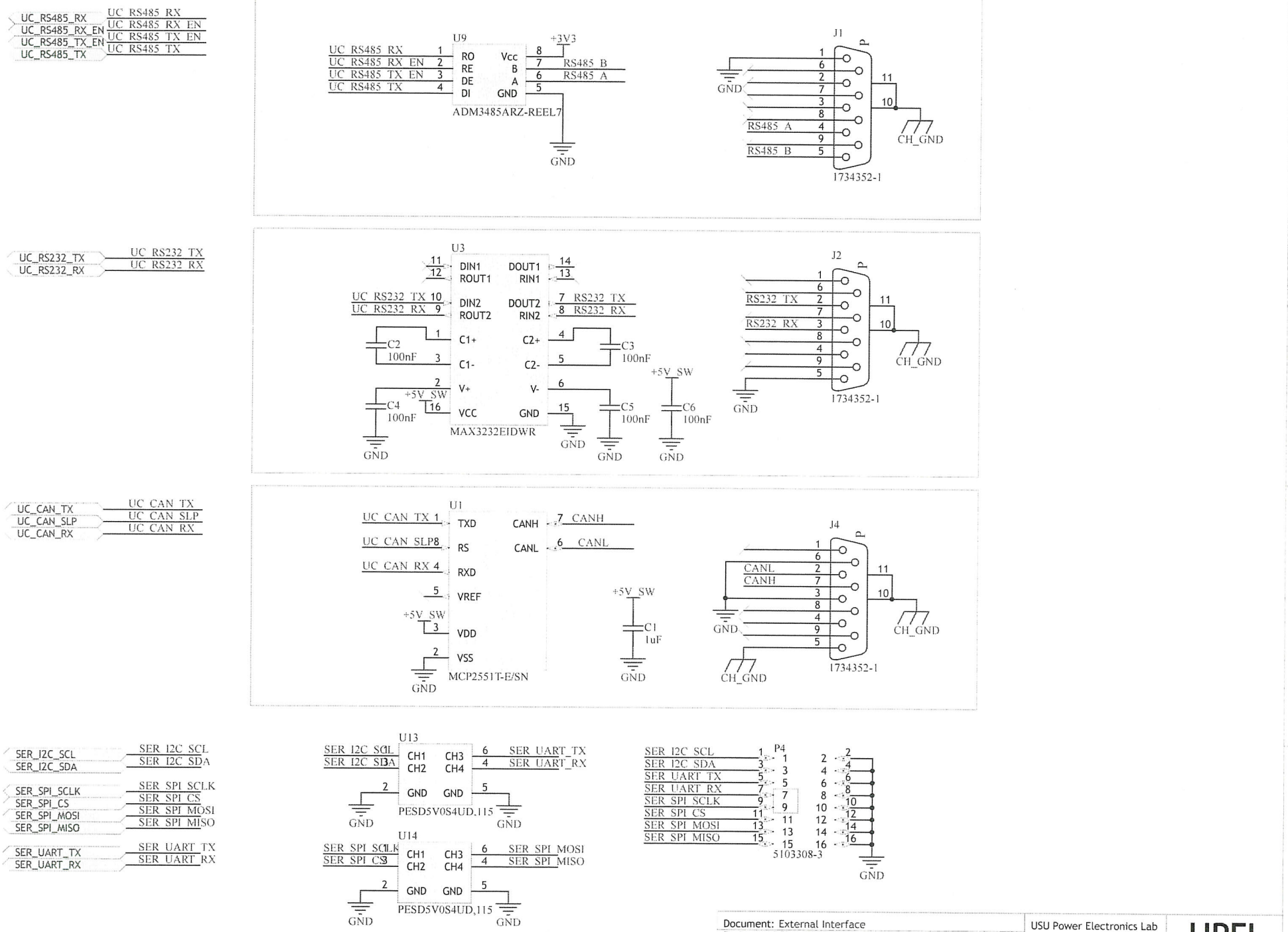
[1] Joëlle; Nigay Laurence Calvary, Gaëlle; Coutaz. From single-user architectural design to pac*: a generic software architecture model for cscw. In Steven Pemberton, editor, *Proceedings of the ACM CHI 97 Human Factors in Computing Systems Conference*, volume March 22-27, pages 242–249. ACM, Atlanta, GA, mar 1997. http://www1.acm.org/sigs/sigchi/chi97/proceedings/paper/jcc.htm.

A. Appendix

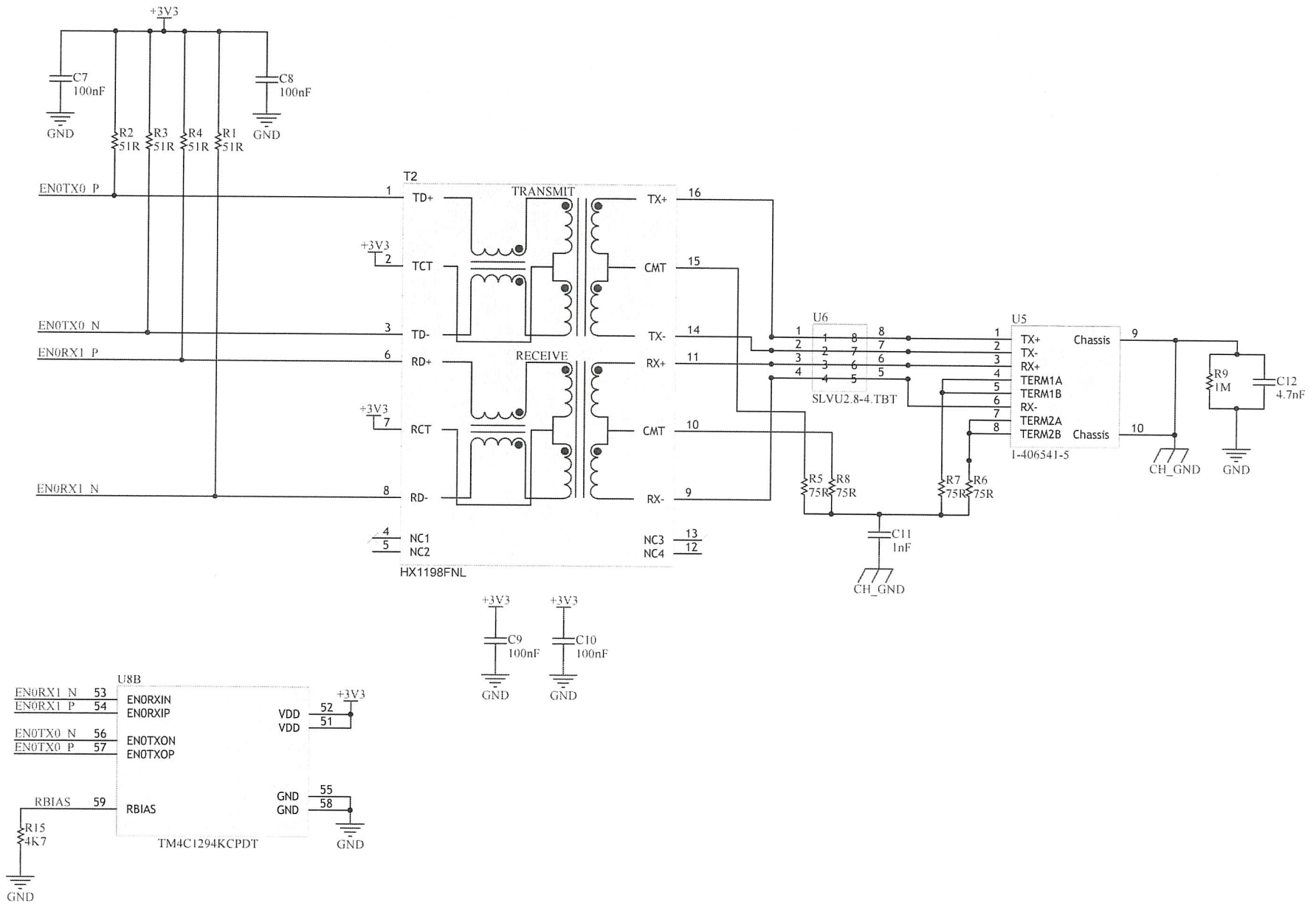
A.1. Hardware Schematics



Document: Power and Microcontroller Support		USU Power Electronics Lab	UPEL
Project: PCB_LACN.PrjPCB		USTAR Building 620, Ste 118	Utah State University Power Electronics Lab
Engineer: KKENNEDY, JJONAS	Revision: v1	1580 N 600 E	
Approved By: *	Date: 4/28/2015	North Logan, UT 84341	
File: C:\Users\Kevin\git\senior-project\Altium\LACN_PCB\Power.SchDoc			Sheet 1 of 5



Document: External Interface	USU Power Electronics Lab
Project: PCB_LACN.PrjPCB	USTAR Building 620, Ste 118
Engineer: KKENNEDY, JJONAS	Revision: v1
Approved By: *	Date: 4/28/2015
File: C:\Users\Kevin\git\senior-project\Altium\LACN_PCB\External_Interface.SchDoc	Sheet 2 of 5



Document: Ethernet Magnetics and Jack		USU Power Electronics Lab
Project: PCB_LACN.PrjPCB		USTAR Building 620, Ste 118
Engineer: KKENNEDY, JJONAS	Revision: v1	1580 N 600 E
Approved By: *	Date: 4/28/2015	North Logan, UT 84341
File: C:\Users\Kevin\git\senior-project\Altium\LACN_PCB\Ethernet_Interface.SchDoc		Sheet 3 of 5

UPEL
Utah State University Power Electronics Lab

From USB_Debug.SchDoc

USB_UART_TX — USB UART TX
USB_UART_RX — USB UART RX
USB_PWR_GOOD — USB PWR GOOD
USB_JTAG_TMS — USB JTAG TMS
USB_JTAG_TDI — USB JTAG TDI
USB_JTAG_TDO — USB JTAG TDO
USB_JTAG_TCK — USB JTAG TCK

From Power.SchDoc

LOAD_ENABLE — LOAD ENABLE

From External_Interface.SchDoc

UC_RS485_RX — UC RS485 RX
UC_RS485_RX_EN — UC RS485 RX EN
UC_RS485_TX_EN — UC RS485 TX EN
UC_RS485_TX — UC RS485 TX
UC_RS232_TX — UC RS232 TX
UC_RS232_RX — UC RS232 RX
UC_CAN_TX — UC CAN TX
UC_CAN_SLP — UC CAN SLP
UC_CAN_RX — UC CAN RX
SER_I2C_SCL — SER I2C SCL
SER_I2C_SDA — SER I2C SDA
SER_SPI_SCLK — SER SPI SCLK
SER_SPI_CS — SER SPI CS
SER_SPI_MOSI — SER SPI MOSI
SER_SPI_MISO — SER SPI MISO
SER_UART_TX — SER UART TX
SER_UART_RX — SER UART RX

U8A

UART0
USB UART TX 33 — PA0/U0RX
USB UART RX 34 — PA1/U0TX
35 — PA2/SSI0CLK
36 — PA3/SSI0FSS
37 — PA4/SSI0XDAT0
38 — PA5/SSI0XDAT1
40 — PA6
41 — PA7

USB JTAG TCK 100 — PC0/TCK/SWCLK
USB JTAG TMS 99 — PC1/TMS/SWDIO
USB JTAG TDI 98 — PC2/TDI
USB JTAG TDO 97 — PC3/TDO/SWO
25 — PC4/C1-
24 — PC5/C1+

UART5
UC RS232 RX 23 — PC6/C0+
UC RS232 TX 22 — PC7/C0-

15 — PE0/AIN3
14 — PE1/AIN2
13 — PE2/AIN1
12 — PE3/AIN0
123 — PE4/AIN9
124 — PE5/AIN8

I2C1
SER I2C SCL 49 — PG0
SER I2C SDA 50 — PG1

116 — PJ0
LCD SER UART TX 117 — PJ1

81 — PL0
82 — PL1
83 — PL2
84 — PL3
85 — PL4
86 — PL5
94 — PL6/USB0DP
93 — PL7/USB0DM

107 — PN0
108 — PN1
109 — PN2
110 — PN3
111 — PN4
112 — PN5

SSI3
SER SPI SCLK 5 — PQ0
SER SPI CS 6 — PQ1
SER SPI MOSI 11 — PQ2
SER SPI MISO 27 — PQ3
102 — PQ4

PB0/USB0ID 95 — UC CAN RX
PB1/USB0VBUS 96 — UC CAN TX
PB2/I2C0SCL 91 — UC CAN SLP
PB3/I2C0SDA 92
PB4/AIN10 121
PB5/AIN11 120

CAN1

PD0/AIN15 1
PD1/AIN14 2
PD2/AIN13 3
PD3/AIN12 4
PD4/AIN7 125
PD5/AIN6 126
PD6/AIN5 127
PD7/AIN4 128

PF0 42
PF1 43
PF2 44
PF3 45
PF4 46

PH0 29 — LOAD ENABLE
PH1 30 — USB PWR GOOD
PH2 31
PH3 32

PK0/AIN16 18 — UC RS485 RX
PK1/AIN17 19 — UC RS485 TX
PK2/AIN18 20 — UC RS485 RX EN
PK3/AIN19 21 — UC RS485 TX EN
PK4 63
PK5 62
PK6 61
PK7 60

UART4

PM0 78
PM1 77
PM2 76
PM3 75
PM4 74
PM5 73
PM6 72
PM7 71

PP0/C2+ 118 — SER UART RX
PP1/C2- 119 — SER UART TX
PP2 103
PP3 104
PP4 105
PP5 106

UART6

TM4C1294KCPDT

Alternate LCD Interface?

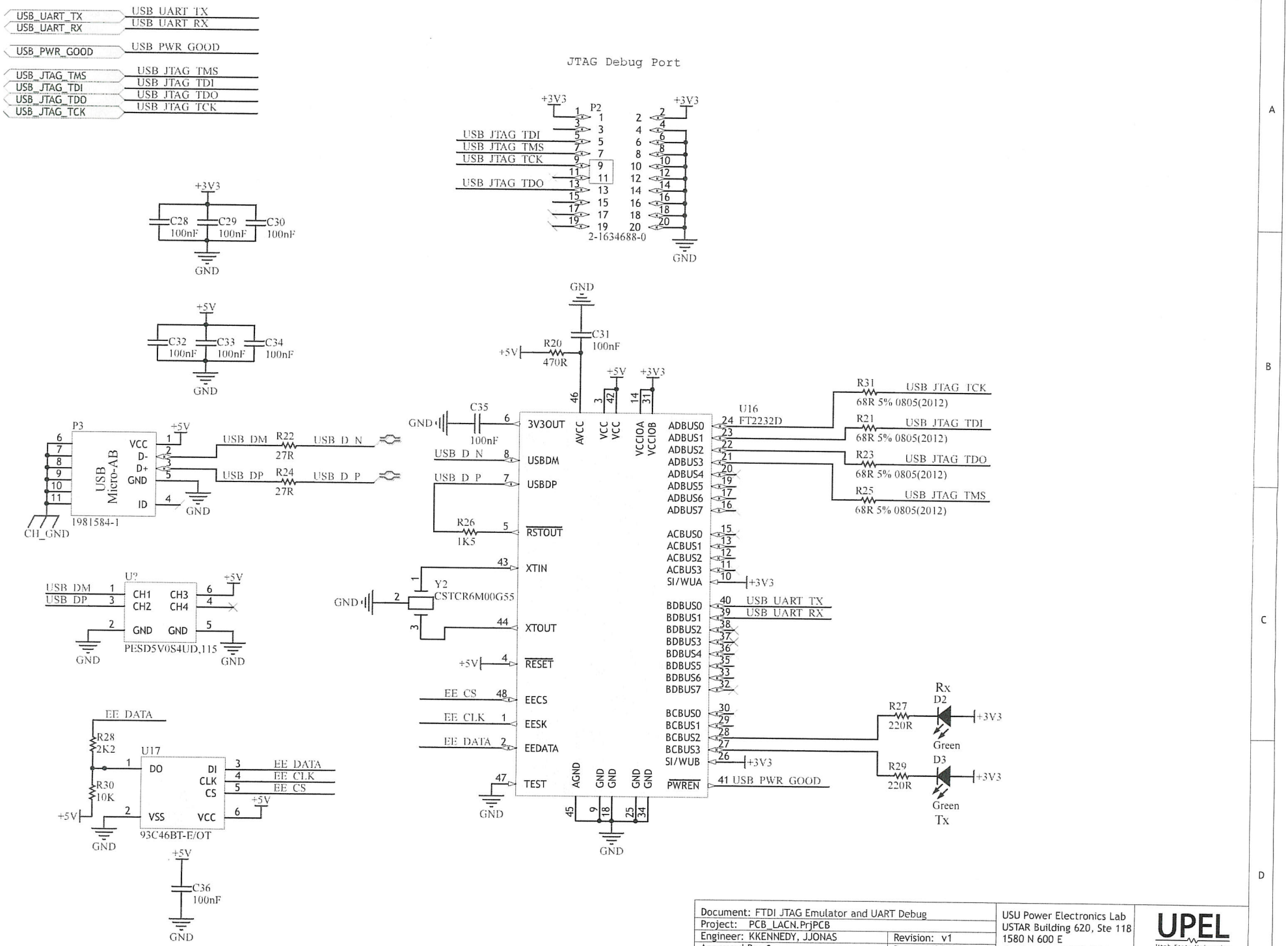
+3V3 — D4 LED0 Green — R14 750R — LED0
+3V3 — D5 LED1 Green — R17 750R — LED1

P1
LCD SER UART TX 1
2
3
GND
+5V
1 O
2 O
3 O
B3B-PH-K-S(LF)(SN)

Mounting

Document: Microcontroller Tiva C TM4C129		USU Power Electronics Lab
Project: PCB_LACN.PrjPCB		USTAR Building 620, Ste 118
Engineer: KKENNEDY, JJONAS	Revision: v1	1580 N 600 E
Approved By: *	Date: 4/28/2015	North Logan, UT 84341
File: C:\Users\Kevin\git\senior-project\Altium\LACN_PCB\Micro_Main.SchDoc		Sheet 4 of 5

UPEL — Utah State University Power Electronics Lab


USB_UART_TX
USB_UART_RX
USB_PWR_GOOD
USB_JTAG_TMS
USB_JTAG_TDI
USB_JTAG_TDO
USB_JTAG_TCK
JTAG Debug Port
P2
2-1634688-0
C28 100nF
C29 100nF
C30 100nF
C32 100nF
C33 100nF
C34 100nF
C31 100nF
C35 100nF
C36 100nF
R20 470R
R22 27R
R24 27R
R26 1K5
R28 2K2
R30 10K
R31 68R 5% 0805(2012)
R21 68R 5% 0805(2012)
R23 68R 5% 0805(2012)
R25 68R 5% 0805(2012)
R27 220R
R29 220R
P3
USB Micro-AB
1981584-1
U16
FT2232D
U?
PESD5V0S4UD,115
U17
93C46BT-E/OT
Y2
CSTCR6M00G55
D2
D3
Green
Rx
Tx
+3V3
+5V
GND
CH_GND
USB_DM
USB_DP
USB_D_N
USB_D_P
EE_DATA
EE_CLK
EE_CS
3V3OUT
USBDM
USBDP
RSTOUT
XTIN
XTOUT
RESET
EECS
EESK
EEDATA
TEST
AVCC
VCC
VCCIOA
VCCIOB
AGND
PWREN
ADBUS0
ADBUS1
ADBUS2
ADBUS3
ADBUS4
ADBUS5
ADBUS6
ADBUS7
ACBUS0
ACBUS1
ACBUS2
ACBUS3
SI/WUA
BDBUS0
BDBUS1
BDBUS2
BDBUS3
BDBUS4
BDBUS5
BDBUS6
BDBUS7
BCBUS0
BCBUS1
BCBUS2
BCBUS3
SI/WUB
USB_PWR_GOOD
Document: FTDI JTAG Emulator and UART Debug
Project: PCB_LACN.PrjPCB
Engineer: KKENNEDY, JJONAS
Revision: v1
Approved By: *
Date: 4/28/2015
File: C:\Users\Kevin\git\senior-project\Altium\LACN_PCB\USB_Debug.SchDoc
USU Power Electronics Lab
USTAR Building 620, Ste 118
1580 N 600 E
North Logan, UT 84341
UPEL
Utah State University Power Electronics Lab
Sheet 5 of 5